Yuhsiang Hong 08/31/2020

Evaluation of Hyperparameter Tunability in Statistical Machine Learning

1. Introduction

Hyperparameters tuning has been an exceptional technique for scientists to optimize the machine learning models. However, different models contain various hyperparameters. The questions of which hyperparameters are suitable to tune or if there is an optimal hyperparameter configuration for each model across all the real-world datasets or if there is a best method to tune hyperparameters have been investigated by many researchers. Therefore, the paper will base on the concept in the reference paper (Probst et al.) and focus on the tunability of hyperparameters and inspecting the methods that is practical and productive in tuning hyperparameters for QSAR datasets.

2. Evaluation Strategy

There are several steps should be done before hyperparameters tuning. First of all, since the dependent variables of QSAR datasets are continuous, both MSE (mean squared error) and RMSE (root-mean-squared error) are simple and clear to compare the performance of different models. Therefore, RMSE will be used as my evaluation criteria. Secondly, applying K-fold cross-validation for each hyperparameter combination can reduce overfitting and selection bias. The mean of the average of K test RMSE across all the datasets will be the evaluation standard for different hyperparameter combinations. Lastly, the hyperparameter combination with the lowest mean of RMSE across datasets will be the optimal hyperparameter combination. Namely, what should be done for each trial in hyperparameter tuning methods is to create different models on different datasets but with the same hyperparameter combination. The objective function is the mean of test RMSE across all the datasets and the goal is to minimize it. The following table is the entire workflow of the simulation study and QSAR datasets.

Simulation Study	For each hyperparameter combination, 1. Build different machine learning models with the same hyperparameter configuration for each dataset 2. Use 5-fold cross-validation for each dataset 3. take the average of 5 test RMSE 4. Calculate the mean of all the average test RMSE across 27 datasets 5. Return all the values of each hyperparameter combination and then select the smallest one as my optimal result
QSAR Datasets	For each hyperparameter combination, 1. Build different machine learning models with the same hyperparameter configuration for each dataset 2. Use 3-fold cross-validation for each dataset 3. Take the average of 3 test RMSE 4. Calculate the mean of all the average test RMSE across 15 QSAR train datasets 5. Return all the mean of RMSE from each hyperparameter combination and then select the smallest one as my optimal result 6. Construct the model from QSAR train datasets using the optimal hyperparameter combination, and test it on QSAR test datasets

Table 1: Workflow. The first row is the workflow of the simulation study and the second row is the workflow of QSAR datasets.

3. Search Space

In this paper, random forest and xgboost (gradient boosting) will be examined for hyperparameters tuning. Both algorithms are very common and widely applied in modern machine learning problems, and also have many hyperparameters that are worth to tune to improve the performance of models. The hyperparameters that are selected to tune in this paper are the almost same as the reference paper (Probst et al.) without **respect.unordered.factors** in random forest and **booster** in xgboost. The reason why is that there are no unordered factors in QSAR datasets, and I will merely focus on tree booster in xgboost. However, the search space in the paper will not be the same as the reference paper (Probst et al.) due to the limitation of time

and hardware devices. Hence, the search space will be smaller, but result might be more practical for people or companies that aim to obtain an optimal result in a short period of time without adequate computing equipment.

3.1. Random Forest

Hyperparameter	Range
n_estimators	{10, 20, …, 990, 1000}. The value is uniformly sampled from the range [10, 1000], and the step of discretization is 10.
bootstrap	{True, False}.
max_samples	{0.1, 0.2, …, 0.9, 0.99}. The value is uniformly sampled from the range [0.1, 1.0], and the step of discretization is 0.1.
max_features	{“sqrt”, “log2”, “auto”, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}.
min_samples_leaf	{1, 0.1, 0.2, 0.3, 0.4, 0.5}.

Table 2: Search space of random forest. The value 0.99 in **max_samples** means that the maximum number of samples drawn from all the observations to train each base estimator is the number of rows. The **max_features** is the number of features to be considered when looking for the best split. The string “auto” means that **max_features** is equal to the number of columns (features) in the dataset. The strings “sqrt” and “log2”, respectively, mean that **max_features** are equal to the square root of the number of columns and the logarithm of the number of columns to the base 2 in the dataset. If **max_features** is a float, then **max_features** is a fraction and the value of **max_features * the number of columns** are the number of columns to be considered at each split. The **min_samples_leaf** is the minimum number of samples required to be at a leaf node. If the value is an integer, it is the minimum number of samples. If the value is a float, it means that the minimum number of samples required to be at a leaf node is equal to the least integer greater than the value of **min_samples_leaf** multiplied by the number of samples. The package used is **RandomForestRegressor** from **sklearn.ensemble** in Python. For more information of hyperparameters, there is an official website of sklearn for users to explore more details (*3.2.4.3.2. sklearn.ensemble.RandomForestRegressor*¶).

3.2. XGBoost

Hyperparameter	Range
num_boost_round	{10, 20, …, 990, 1000}. The value is uniformly sampled from the range [10, 1000], and the step of discretization is 10.
lambda	{2^(-9), 2^(-8), …, 2^(9), 2^(10)}. The value is discretely uniformly sampled on a geometric sequence which has the range from 2^(-9) to 2^(10), and the common ratio is 2.
alpha	It is the same as **lambda**.
eta	{0, 0.1, …, 0.9, 1.0}. The value is uniformly sampled from the range [0, 1.0], and the step of discretization is 0.1.
subsample	{0.1, 0.2, …, 0.9, 1.0}. The value is uniformly sampled from the range [0.1, 1.0], and the step of discretization is 0.1.
max_depth	{1, 2, …, 14, 15}. The value is uniformly sampled from the range [1, 15], and the step of discretization is 1.
min_child_weight	{1, 2, 4, 8, 16, 32, 64, 128}.
colsample_bytree	It is the same as **subsample**.
colsample_bylevel	It is the same as **subsample**.

Table 3: Search space of xgboost. The setup in xgboost is mostly the same as the reference paper. The package used is **xgboost** in Python. For more information of hyperparameters, there is an official website of xgboost for users to explore more details (*Python API Reference¶).*

4. Simulation Study

Simulation study is a good way to acknowledge the effect of different hyperparameter tuning methods on simulated data before analyzing real-world data. In addition, it can also provide an overview of which hyperparameters might be critical in the tuning process and set up the workflow for the research on real-world data.

There are two parts in the simulation study. The first part is checking the tunability of a single hyperparameter, in other words, tuning one hyperparameter at a time. The second part concentrates on the effect of tuning all the hyperparameters together using three different tuning methods, which are Grid Search, Random Search, and Bayesian Optimization. Furthermore, I

will compare the improvement of the mean of RMSE across datasets and the time-efficiency after obtaining the optimal hyperparameter combinations using three different tuning methods.

4.1 Datasets

The simulated datasets are generated by the package **make_friedman1** from **sklearn.datasets**. There are 27 datasets in total with different numbers of rows, numbers of columns, and noises, which are displayed in Table 4. Moreover, in order to attain an optimal result in a feasible time, 5-fold cross-validation is implemented in the simulation study.

Number of rows	100, 500, 1000
Number of columns	100, 500, 1000
Noise	1, 5, 10

Table 4: Friedman1 Datasets.

4.2. Tuning a Specific Hyperparameter

Most of the grid points that are used in the tuning process are the same as the points in the search space. However, for **n_estimators** in random forest, I only select values from 10 to 500 because of the limitation of time. The other difference is that for **max_features**, I will only use "auto", "sqrt", and "log2" instead of all the values in Table 2 in the entire simulation study with the same reason. Table 5 presents the optimal result from tuning each hyperparameter.

Hyperparameter (Random Forest)	Optimal Result	Hyperparameter (XGBoost)	Optimal Result
n_estimators	470	num_boost_round	20
bootstrap	False	lambda	4
max_samples	1 (default)	alpha	0.0625
max_features	"auto"	eta	0.3 (default)
min_samples_leaf	1 (default)	subsample	1 (default)
		max_depth	2
		min_child_weight	64

colsample_bytree	1 (default)
colsample_bylevel	0.3

Table 5: Optimal Results of a Specific Hyperparameter. The first two columns are the optimal results of random forest, and the last two columns are the optimal results of xgboost. There are some cells which have the word, default, behind the value. This means that the optimal result of the hyperparameter is equal to the default value of the hyperparameter.

Hyperparameter (Random Forest)	Mean of RMSE	Improvement Rate (%)
default	7.1884	-
n_estimators	7.1581	0.42%
bootstrap	7.1334	0.77%
max_samples	7.1865	0.04%
max_features	6.5972	8.22%
min_samples_leaf	7.1620	0.37%
Hyperparameter (XGBoost)	Mean of RMSE	Improvement Rate (%)
default	6.9601	-
num_boost_round	6.9358	0.35%
lambda	6.9470	0.19%
alpha	6.9312	0.42%
eta	6.9601	-
subsample	6.9601	-
max_depth	6.6491	4.47%
min_child_weight	6.6303	4.74%
colsample_bytree	6.9601	-
colsample_bylevel	6.9313	0.41%

Table 6: Mean of RMSE Using Optimal Results. The second and ninth rows in the table are the mean of the RMSE using default values. The reason why they appear in this table is for the comparison between mean of RMSE using default and optimal values. The improvement rate for

each hyperparameter in the third column is calculated by **(mean of RMSE using default values – mean of RMSE using optimal values)/mean of RMSE using default values**.

In random forest, most of the hyperparameters don't show significant improvement from tuning, except for **max_features**. As we can see from Table 6, the improvement rate is over 8%. The remaining hyperparameters don't have the improvement rate over 1%, so it is quite large, and the result indicates that **max_features** might be worth tuning and should be kept under observation in further analysis. The **bootstrap** also shows little improvement, which has the second largest improvement rate among all the hyperparameters, but compared with **max_features**, it doesn't have the same level of improvement as **max_features** does.

In xgboost, **max_depth** and **min_child_weight** show larger improvement rate than other hyperparameters do, which implies that these two hyperparameter might also be worth tuning.

4.3. Tuning All Hyperparameters

There are three common ways to do hyperparameter optimization, which are grid search, random search, and Bayesian Optimization. Grid search inspects through all the point in the search space to obtain the optimal result. On the other hand, random search examines certain numbers of independent trials in the search space and chooses the optimal results from these trials. Bayesian optimization is a sequential design technique for global optimization of black-box functions that does not assume any functional forms. Basically, it incorporates prior belief about the objective function, which is the mean of RMSE in this project, and updates the prior with samples drawn from the objective function to get a posterior that better approximates the objective function. The model used for approximating the objective function is called surrogate model. One of the most common surrogate models is Tree Parzen Estimators (TPE). It is the default surrogate model for most of the Bayesian Optimization packages in R or Python. Therefore, this paper will implement TPE in Bayesian Optimization.

4.3.1. Random Forest

Since grid search examines all the points in the search space, it is not possible for me to investigate every point in the search space. Therefore, the idea of choosing the points is to scatter

them on the grid as uniformly and widely as possible. Table 7 displays all the grid points for each hyperparameter.

Hyperparameter (Random Forest)	Grid Point
n_estimators	[10, 100, 300, 500, 700, 1000]
bootstrap	[True, False]
max_samples	[0.1, 0.5, 1.0]
max_features	[“sqrt”, “auto”, “log2”]
min_samples_leaf	[1, 0.1, 0.3, 0.5]

Table 7: Grid Points of Random Forest in Grid Search. Total number of grid points is 288.

Table 8 displays that the search space of random search and Bayesian optimization, which is basically the same as the search space presented in Table 2.

Hyperparameter (Random Forest)	Grid Point
n_estimators	[10, 100, 300, 500, 700, 1000]
bootstrap	[True, False]
max_samples	[0.1, 0.2, 0.4, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99]
max_features	[“sqrt”, “auto”, “log2”]
min_samples_leaf	[1, 0.1, 0.2, 0.3, 0.4, 0.5]

Table 8: Points in the Search Space of Random Search and Bayesian Optimization. Each method uses 100 trials.

Table 9 shows the optimal results of random forest using three methods and Table 10 compares the results of three methods. From the improvement rate in Table 10, grid search has the smallest improvement and random search and Bayesian optimization both improve significantly.

Grid Search	number 132 value 6.73054 datetime_start 2020-07-24 08:45:54.178549 datetime_complete 2020-07-24 09:25:45.755663 duration 0 days 00:39:51.577114000 params_bootstrap True params_max_features auto params_max_samples 0.99 params_min_samples_leaf 1 params_n_estimators 1000 system_attrs_grid_id 335 system_attrs_search_space OrderedDict([('bootstrap', [False, True]), ('m... state COMPLETE Name: 132, dtype: object
Random Search	number 91 value 6.57085 datetime_start 2020-07-24 20:28:31.945827 datetime_complete 2020-07-24 20:46:56.664552 duration 0 days 00:18:24.718725000 params_bootstrap True params_max_features auto params_max_samples 0.9 params_min_samples_leaf 1 params_n_estimators 1000 state COMPLETE Name: 91, dtype: object
Bayesian Optimization	number 89 value 6.59018 datetime_start 2020-07-25 15:15:51.100678 datetime_complete 2020-07-25 15:23:45.914445 duration 0 days 00:07:54.813767000 params_bootstrap True params_max_features auto params_max_samples 0.7 params_min_samples_leaf 1 params_n_estimators 500 state COMPLETE Name: 89, dtype: object

Table 9: Optimal results of Random Forest Implementing Three Methods.

Tuning Method	Mean of RMSE	Improvement Rate (%)
Grid Search	6.7305	6.37%
Random Search	6.5709	8.59%
Bayesian Optimization	6.5902	8.32%

Table 10: Mean of RMSE from Optimal Results Using Three Methods for Random Forest. The improvement rates in the table are calculated the same way as in Table 6.

4.3.2. XGBoost

In grid search, tuning every point in the search space presented in Table 3 is impossible for me to finish in a reasonable time. Therefore, I use the same concept that I use in random forest to select grid points from the search space. In addition, I drop three hyperparameters from tuning, which are **eta**, **max_samples**, and **colsample_bytree** because it will take a lot of time to tune all the hyperparameters in xgboost and the reason why I choose these three is because their optimal results from tuning one hyperparameter at a time are the same as the default values. Table 11 displays all the grid points for each hyperparameter.

Hyperparameter (XGBoost)	Grid Point
num_boost_round	[10, 100, 500]
lambda	[0.25, 1, 4]
alpha	[0, 0.625, 1]
max_depth	[2, 6, 10]
min_child_weight	[1, 16, 64]
colsample_bylevel	[0.3, 1]

Table 11: Grid Points of XGBoost in Grid Search. Total number of grid points is 486.

Table 12 displays that the search space of random search and Bayesian optimization, which is basically the same as the search space presented in Table 3.

Hyperparameter (XGBoost)	Grid Point
num_boost_round	[10, 20, …, 490, 500]
lambda	[2^(-9), 2^(-8), …, 2^(9), 2^(10)]
alpha	[2^(-9), 2^(-8), …, 2^(9), 2^(10)]
eta	[0, 0.1, 0.2, …, 0.9, 1.0]
subsample	[0.1, 0.2, …, 0.9, 1.0]
max_depth	[1, 2, …, 14, 15]
min_child_weight	[1, 2, 4, 8, 16, 32, 64, 128]
colsample_bytree	[0.1, 0.2, …, 0.9, 1.0]

colsample_bylevel	[0.1, 0.2, …, 0.9, 1.0]

Table 12: Points in the Search Space of Random Search and Bayesian Optimization. Each method uses 100 trials.

Table 13 shows the optimal results of xgboost using three methods and Table 14 compares the results of three methods. From the improvement rate in Table 14, grid search shows the smallest improvement like it does in random forest, and Bayesian optimization this time has the highest improvement rate.

Grid Search	number 137 value 6.6243 datetime_start 2020-07-24 06:41:10.704274 datetime_complete 2020-07-24 06:41:14.650843 duration 0 days 00:00:03.946569000 params_alpha 1 params_colsample_bytree 1 params_lambda 1 params_max_depth 10 params_min_child_weight 64 params_num_boost_round 10 system_attrs_grid_id 456 system_attrs_search_space OrderedDict([('alpha', [0, 0.0625, 1]), ('boos... state COMPLETE Name: 137, dtype: object
Random Search	number 81 value 6.61093 datetime_start 2020-07-25 02:25:24.526967 datetime_complete 2020-07-25 02:26:12.504942 duration 0 days 00:00:47.977975000 params_alpha 0 params_colsample_bylevel 0.1 params_colsample_bytree 0.9 params_eta 0.2 params_lambda 1024 params_max_depth 10 params_min_child_weight 32 params_num_boost_round 440 params_subsample 0.9 state COMPLETE Name: 81, dtype: object

Bayesian Optimization	number 99 value 6.50044 datetime_start 2020-07-25 04:04:29.370426 datetime_complete 2020-07-25 04:04:52.142324 duration 0 days 00:00:22.771898000 params_alpha 4 params_colsample_bylevel 0.2 params_colsample_bytree 1 params_eta 0.1 params_lambda 256 params_max_depth 1 params_min_child_weight 8 params_num_boost_round 390 params_subsample 1 state COMPLETE Name: 99, dtype: object

Table 13: Optimal results of XGBoost Implementing Three Methods. There is a typo in grid search. It should be params_colsample_bylevel instead of params_colsample_bytree. Other than that, everything is fine.

Tuning Method	Mean of RMSE	Improvement Rate (%)
Grid Search	6.6243	4.82%
Random Search	6.6109	5.02%
Bayesian Optimization	6.5004	6.60%

Table 14: Mean of RMSE from Optimal Results Using Three Methods for XGBoost. The improvement rates are calculated the same way as in Table 6.

5. QSAR Datasets

The workflow in QSAR datasets is basically the same as the simulation study. However, there are still several differences between the simulation study and QSAR datasets. The first difference is the datasets. The total number of the datasets in QSAR datasets is 15, but each dataset is a lot bigger than the simulated datasets. Thus, instead of 5-fold cross-validation used in simulation study, 3-fold cross-validation will be implemented in QSAR datasets. The other thing is that after obtaining the optimal results from tuning, I will construct the model from QSAR train datasets using the optimal hyperparameter combination and test it on QSAR test datasets.

5.1 Tuning a Specific Hyperparameter

Most of the grid points that are used in the tuning process are the same as the points in the search space in Table 2 and Table 3, except for **n_estimators** in random forest and **num_boost_round** in xgboost. For **n_estimators**, since the datasets are extremely large, the values are set from 10 to 100 with the step of discretization equal to 10 and from 100 to 1000 with the step of discretization equal to 100. For **num_boost_round**, it takes over an hour to tune any hyperparameter combinations with **num_boost_round** over 100, so it will not be attainable to set the value over 100 and finish the tuning process in a limited time. Table 15 presents the optimal result from tuning each hyperparameter.

Hyperparameter (Random Forest)	Optimal Result	Hyperparameter (XGBoost)	Optimal Result
n_estimators	800	num_boost_round	100
bootstrap	False	lambda	0.125
max_samples	1 (default)	alpha	1.0
max_features	0.5	eta	0.4
min_samples_leaf	1 (default)	subsample	1.0 (default)
		max_depth	7
		min_child_weight	4
		colsample_bytree	0.8
		colsample_bylevel	0.8

Table 15: Optimal Results of a Specific Hyperparameter. The first two columns are the optimal results of random forest, and the last two columns are the optimal results of xgboost. There are some cells which have the word, default, behind the value. This means that the optimal results of the hyperparameters are equal to the default values of the hyperparameters.

Hyperparameter (Random Forest)	Mean of RMSE	Improvement Rate (%)
default	1.9652	-
n_estimators	1.9503	0.76%
bootstrap	1.9184	2.38%
max_samples	1.9652	-
max_features	1.9288	1.85%
min_samples_leaf	1.9652	-
Hyperparameter (XGBoost)	Mean of RMSE	Improvement Rate (%)
default	2.0738	-
num_boost_round	1.9442	6.25%
lambda	2.0684	0.26%
alpha	2.0574	0.79%
eta	2.0520	1.05%
subsample	2.0738	-
max_depth	2.0505	1.12%
min_child_weight	2.0600	0.67%
colsample_bytree	2.0586	0.73%
colsample_bylevel	2.0718	0.10%

Table 16: Mean of RMSE Using Optimal Results. The second and ninth rows in the table are the mean of the RMSE using default values. The improvement rate for each hyperparameter in the third column is calculated by **(mean of RMSE using default values – mean of RMSE using optimal values)/mean of RMSE using default values**.

In random forest, most hyperparameters don't show significant improvement from tuning, except for **bootstrap** and **max_features**. As we can see from Table 16, both improvement rates are over 1%. This result also aligns with the result in the simulation study. These two hyperparameters exhibit the largest improvement among all the hyperparameters in both studies.

In xgboost, **num_boost_round**, **eta**, and **max_depth** show the largest improvement rate among all the hyperparameters. Hence, these hyperparameters might be valuable to tune. The optimal value of **num_boost_round** is on the boundary of the tuning range, which implies the tuning range of this hyperparameter should be larger. Nevertheless, this problem is not able to be solved in this paper, but it is worth to be improved for the future research.

5.2 Tuning All Hyperparameters

Tuning all hyperparameters together at the same time is very time-consuming, especially when the datasets are huge. In simulation study, the time required to achieve the optimal results with any tuning method is already massive. The minimum time to complete random search in simulation study is more than four hours. Accordingly, withdrawing certain hyperparameters and modifying the ranges of hyperparameters are essential to attain the optimal results.

5.2.1 Random Forest

There is a problem when I use random forest on QSAR datasets. When the **min_samples_leaf** is set to one if integer or lower than 0.1 if float, which means that the minimum number of samples required to be at a leaf node is less than **0.1*numbers of samples**, the execution time of tuning one hyperparameter combination is more than 1.5 hour. Hence, instead of using **min_samples_leaf** equal to 1 as the default value, I remove 1 from the search space and use 0.1 as my default values. Afterwards, I compare the tuning results to the new default values. In addition, since **max_features** equal to "log2" doesn't show any improvement from previous analysis, only "auto" and "sqrt" are used in this time.

Besides the modification of tuning **min_samples_leaf**, the process and notion of tuning other hyperparameters remain unchanged from the simulation study for three methods.

Hyperparameter (Random Forest)	Grid Point
n_estimators	[10, 100, 300, 500, 700, 1000]
bootstrap	[True, False]
max_samples	[0.1, 0.5, 0.99]
max_features	["sqrt", "auto"]
min_samples_leaf	[0.1, 0.3, 0.5]

Table 17: Grid Points of Random Forest in Grid Search. Total number of grid points is 144.

Table 18 displays that the search space of random search and Bayesian optimization, which is basically the same as the search space presented in Table 2. Because **max_features** shows improvement in the simulation study and the research of tuning one hyperparameter in QSAR datasets, I will use all the point in its range.

Hyperparameter (Random Forest)	Grid Point
n_estimators	[10, 100, 200, …, 900, 1000]
bootstrap	[True, False]
max_samples	[0.1, 0.2, …, 0.8, 0.9, 0.99]
max_features	[“sqrt”, “auto”, 0.1, 0.2, …, 0.8, 0.9]
min_samples_leaf	[0.1, 0.2, 0.3, 0.4, 0.5]

Table 18: Points in the Search Space of Random Search and Bayesian Optimization. Total numbers of points in the search space is 6655 and each method uses 50 trials.

Table 19 presents the optimal results of random forest using three methods and Table 20 compares the optimal results using three methods. Based on the improvement rate in Table 20, grid search has the least improvement and Bayesian optimization has the largest improvement rate. Moreover, the execution time of grid search is longer than other two methods.

Grid Search	number 56 value 2.57515 datetime_start 2020-08-19 07:16:40.524520 datetime_complete 2020-08-19 07:36:45.741477 duration 0 days 00:20:05.216957000 params_bootstrap True params_max_features auto params_max_samples 0.99 params_min_samples_leaf 0.1 params_n_estimators 700 system_attrs_grid_id 148 system_attrs_search_space OrderedDict([('bootstrap', [False, True]), ('m... state COMPLETE Name: 56, dtype: object

Random Search	number 31 value 2.52189 datetime_start 2020-08-15 05:42:24.029562 datetime_complete 2020-08-15 05:46:22.031993 duration 0 days 00:03:58.002431000 params_bootstrap False params_max_features 0.1 params_max_samples NaN params_min_samples_leaf 0.1 params_n_estimators 400 state COMPLETE Name: 31, dtype: object
Bayesian Optimization	number 21 value 2.48504 datetime_start 2020-08-14 20:56:28.528580 datetime_complete 2020-08-14 21:13:08.490926 duration 0 days 00:16:39.962346000 params_bootstrap False params_max_features 0.3 params_max_samples NaN params_min_samples_leaf 0.1 params_n_estimators 900 state COMPLETE Name: 21, dtype: object

Table 19: Optimal results of Random Forest Implementing Three Methods.

Tuning Method	Mean of RMSE (Default: 2.9241)	Improvement Rate (%)	Execution Time (hour: minute)
Grid Search	2.5751	11.94%	12:29
Random Search	2.5219	13.75%	4:06
Bayesian Optimization	2.4850	15.02%	8:53

Table 20: Comparison of Optimal Results Using Three Methods for Random Forest. The improvement rates in this table are calculated the same way as in Table 6. The value, 2.9241, is the mean of RMSE with all the hyperparameters in random forest using default values, except for **min_samples_leaf** which is equal to 0.1.

Let's test the optimal hyperparameter combinations on testing datasets and see if the improvement remains similar. Table 21 shows that although each improvement rate decreases, random search and Bayesian optimization still have larger improvement rates than grid search.

Method	Mean of RMSE (Default: 3.2153)	Improvement Rate (%)
Grid Search	2.9470	8.35%
Random Search	2.8782	10.48%
Bayesian Optimization	2.8665	10.85%

Table 21: Comparison of the Results of Using Optimal Hyperparameter Combinations on QSAR Testing Datasets. The value, 3.2153, is the mean of RMSE with all the hyperparameters in random forest using default values, except for **min_samples_leaf** equal to 0.1.

5.2.2 XGBoost

In grid search, due to the fact that tuning **num_boost_round** takes the most time, it is not feasible to tune this hyperparameter. Furthermore, I will only tune **eta**, **max_depth**, and **min_child_weight** in grid search because for the first two hyperparameters, they display the greatest improvement from tuning themselves alone. The reason why I also include **min_child_weight** for tuning is that **max_depth** and **min_child_weight** are the hyperparameters that determine the complexity of the tree in xgboost, and theoretically, when the **max_depth** increases, **min_child_weight** should also increase to avoid overfitting. The concept of choosing the points for tuning is the same as the concept implemented in the simulation study. The search space of grid search is presented in Table 22.

Hyperparameter (xgboost)	Grid Point
eta	[0.1, 0.2, …, 0.8, 0.9, 1.0]
max_depth	[6, 10, 15]
min_child_weight	[1, 16, 64, 128]

Table 22: Grid Points of XGBoost in Grid Search. Total number of grid points is 120.

Table 23 displays that the search space of random search and Bayesian optimization. As the reason stated previously, the values of **num_boost_round** is set from 10 to 100 with the step of discretization equal to 10.

Hyperparameter (XGBoost)	Grid Point
num_boost_round	[10, 20, …, 90, 100]
lambda	[2^(-9), 2^(-8), …, 2^(9), 2^(10)]
alpha	[2^(-9), 2^(-8), …, 2^(9), 2^(10)]
eta	[0, 0.1, 0.2, …, 0.9, 1.0]
subsample	[0.1, 0.2, …, 0.9, 1.0]
max_depth	[1, 2, …, 14, 15]
min_child_weight	[1, 2, 4, 8, 16, 32, 64, 128]
colsample_bytree	[0.1, 0.2, …, 0.9, 1.0]
colsample_bylevel	[0.1, 0.2, …, 0.9, 1.0]

Table 23: Points in the Search Space of Random Search and Bayesian Optimization. Total numbers of points in the search space is more than 5 billion and each method uses 50 trials.

Table 24 presents the optimal results of xgboost using three methods and Table 25 compares the optimal results. Based on the improvement rate in Table 25, grid search has the smallest improvement and Bayesian optimization has the largest improvement rate. However, the execution time of grid search this time is the lowest because num_boost_round is not tuned in grid search, so it is set to be the default value, 10, which is relatively small. Consequently, it has the smallest execution time even though there are 120 grid points in the search space.

Grid Search	number 62 value 2.03679 datetime_start 2020-08-20 05:10:58.278545 datetime_complete 2020-08-20 05:19:38.329354 duration 0 days 00:08:40.050809000 params_eta 0.3 params_max_depth 15 params_min_child_weight 16 system_attrs_grid_id 33 system_attrs_search_space OrderedDict([('eta', [0.1, 0.2, 0.3, 0.4, 0.5,... state COMPLETE Name: 62, dtype: object

Random Search	number 22 value 1.95176 datetime_start 2020-08-18 16:29:44.287506 datetime_complete 2020-08-18 16:53:03.435307 duration 0 days 00:23:19.147801000 params_alpha 0.015625 params_colsample_bylevel 0.6 params_colsample_bytree 0.9 params_eta 0.1 params_lambda 0.00195312 params_max_depth 7 params_min_child_weight 8 params_num_boost_round 60 params_subsample 0.7 state COMPLETE Name: 22, dtype: object
Bayesian Optimization	number 38 value 1.90901 datetime_start 2020-08-17 13:09:08.468815 datetime_complete 2020-08-17 14:11:51.776328 duration 0 days 01:02:43.307513000 params_alpha 0.03125 params_colsample_bylevel 0.8 params_colsample_bytree 1 params_eta 0.1 params_lambda 16 params_max_depth 11 params_min_child_weight 4 params_num_boost_round 80 params_subsample 0.7 state COMPLETE Name: 38, dtype: object

Table 24: Optimal results of XGBoost Implementing Three Methods.

Tuning Method	Mean of RMSE (Default: 2.0738)	Improvement Rate (%)	Execution Time (hour: minute)
Grid Search	2.0368	1.78 %	14:06
Random Search	1.9518	5.88 %	18:52
Bayesian Optimization	1.9090	7.95 %	23:28

Table 25: Comparison of Optimal Results Using Three Methods for XGBoost. The improvement rates in this table are calculated the same way as in Table 6. The value, 2.0738, is the mean of RMSE with all the hyperparameters in xgboost using default values.

Let's test the optimal hyperparameter combinations on testing datasets and see if the improvement remains similar. Table 26 shows that the improvement rates of Bayesian optimization and random search again have the larger rates than grid search.

As we can see from Table 21 and 26, the improvement rates from random forest are larger than xgboost. This phenomenon comes from the fact that the real default values in random forest are not used as mentioned previously. As the result, the space of the improvement on random forest should also be greater than xgboost.

Method	Mean of RMSE (Default: 2.5955)	Improvement Rate (%)
Grid Search	2.5527	1.65%
Random Search	2.3938	7.77%
Bayesian Optimization	2.4705	4.82%

Table 26: Comparison of the Results of Using Optimal Hyperparameter Combinations on QSAR Testing Datasets. The value, 2.5955, is the mean of RMSE with all the hyperparameters in xgboost using default values.

6. Conclusion and Improvement

Each method has its pros and cons. Grid search can be used if we have some deep knowledge in the specific machine learning algorithms and datasets. Random search does not guarantee to have some improvement to the model since it is randomly selected points from the search space. On the other hand, Bayesian optimization can get the result that improves the model, but it needs more execution time than Random Search.

From the results of the simulation study and QSAR datasets, Bayesian optimization and random search have similar improvement rates. In some scenarios, Bayesian optimization have larger rates, and in other scenarios, it doesn't. However, both methods show greater improvement than grid search does. In the aspect of time efficiency, random search, on average, has the lowest running time to obtain the optimal result. As the result, for QSAR datasets, random search is more preferable for me to use for hyperparameters tuning, especially when we use a certain amount of time, random search can examine more trials than Bayesian optimization, which leads to a higher possibility to achieve a better result.

For the future improvement, first of all, I would like to increase the points in the search space of three methods, especially grid search. The results of grid search show the least improvement, but if I were able to try more points in the search space, the improvement rate could be the greatest. However, it also needs more time or a better computing equipment to achieve.

Secondly, some hyperparameters should be tuned with other hyperparameters because of their dependency for each other. For instance, in xgboost, the smaller learning rate (**eta)** generally requires more trees (**num_boost_round**) to be added to the model. In the future approaches, this kind of relationship should also be considered while we tune the hyperparameters.

7. Software Recommendation

I personally recommend Python over R for hyperparameters tuning because the online community of Python is larger, so more information and details could be found on the internet. For the package to implement in hyperparameters tuning, I recommend Optuna for several reasons. First, it is easy to use, and the coding is really straightforward. Second, we can do grid search, random search, and Bayesian optimization from this package without switching to others. Last but not least, there is an official website of Optuna that explains all the basic concepts and presents you where to find more information.

8. References

Probst, Philipp, et al. "Tunability: Importance of Hyperparameters of Machine Learning Algorithms." *ArXiv.org*, 22 Oct. 2018, arxiv.org/abs/1802.09596.

3.2.4.3.2. Sklearn.ensemble.RandomForestRegressor¶. scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

Python API Reference¶. (n.d.). Retrieved August 31, 2020, from https://xgboost.readthedocs.io/en/latest/python/python_api.html